REMARKS/ARGUMENTS

Pursuant to the requirement of 37 CFR 1.121(b), and as stated above, please substitute and replace all the claim sheets, as amended and as originally filed, with the above amended set of claims.

The following claim rejections and objections were noted from the Office Action dated April 9, 2003, and pursuant to each paragraph, presented in the same order, arguments follow.

Claim Rejections – 35 USC § 112

2. ***Claims 3-8, 11 were rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention.***

In response to this rejection, Claims 3-8 and 11 were amended to overcome the 112 rejections in that the terms "computer screen template", "various value display windows", and others have been changed to more clearly define the invention.

However, one of the terms that were objected to, i.e. "a new program is written or created", has been left in the claim because that is what the invention truly does. This type of program is called a "compiler" (see attached Merriam Webster dictionary Internet definition). Compilers are computer programs that write new programs depending upon certain values entered, and they are quite common in the computing field. Also enclosed are copies of the Borland® Developer Network Internet instructions regarding the way that an executable file in Linux or Win32 can spring a "loader" into action for loading the file, mapping it to a memory, and initiating the execution of an executable initial entry-point.

Careful review of the originally filed computer software will reveal an executable file of this type that responds to custom data. In other words, the use of Linux lets us write self-modifying executables. In the present invention, that means that the present invention can write its own new computer programs by the addition of custom data. Hence, a new computer program can be written, *by the addition of custom data into the computer itself,* to run any one of the wheels of the presently claimed invention in a desired pattern. All of this can be achieved *without* the operator having to program the computer himself. The computer will write its own new program.

Of course, Applicants do not claim to have invented CNC machines. However, they can only guess what type of computer numerically controlled devices are utilized in the cited prior art. The cited prior art do not teach, suggest nor disclose any computers that write their own programs. In that regard, Applicants are herewith enclosing a copy of the "new way" and the "old way" illustrations to show that the "old way" of programming a CNC machine was to use something called "G Code". Note that such code cannot be modified for the values on the screen. The entire computer program has to be reprogrammed, usually taking hours of the time of a trained professional.

Further to show the contrast between the present invention and the cited prior art, an additional sheet of illustrations has been included, in which a pop-up value box display shows a value of 00.49000. In direct contradistinction to the programming required by the cited prior art, the present invention allows for the customization of individual data points, followed by the writing of a new program, pursuant to the customization of the data by the computer.

Applicants sincerely hope that this explanation is sufficient to explain the patentably distinct differences between the present invention and the cited prior art, and therefore, they respectfully submit that the claims, as are they are currently written, should be allowable, and thereby request such an allowance.

Claim Rejections – 35 USC § 102

4. Claims 3-8, 11, as best understood, were rejected under 35 U.S.C. 102(e) as being anticipated by Maack-5,766,057.

In response to this rejection, Applicants would like to point out the patentably distinct structural differences between the present invention and the cited Maack reference. Most notably, we need to look at column 2, lines 27-28 and lines 31-32 of Maack to see that there are two separate feed axes disclosed and claimed. In direct contradistinction, the present invention discloses a single axis for the grinding wheel and the regulating wheel, whereby the accuracy of the overall grinding operation is greatly increased to new levels as the wheels are on the same axis, so that differences in shifting the two wheels between axes is eliminated, rendering a more accurate grinding job.

This significant structural difference, when combined with the new computer programming capabilities, renders the present invention patentably distinct over the cited prior art sufficiently for the Applicants to respectfully request an allowance for all claims being considered.

Response to Arguments

For the reasons above, Applicants respectfully submit that claims 3-8 and 11 are now in condition for allowance, and request that the Examiner give such an allowance.

Applicants wish to thank the Examiner for her thorough examination, and hope, that by these explanations and Amendments, the subject matter of the present invention is now more clearly stated, such that a closer review of the present invention, in light of the amendments and arguments made here, will give solid support for an allowance. Consequently, Applicants request

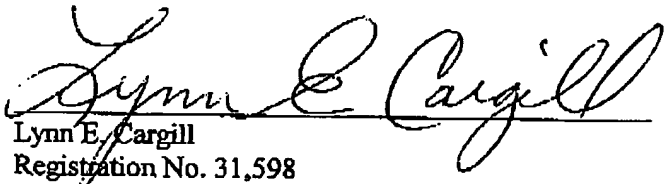
reconsideration in the instant Application and withdrawal of all grounds of rejection and objection in view of the amendments and the following discussion.

If the Examiner feels that the prosecution of this Application can be expedited by conversation, she is courteously requested to place a telephone call to Applicants' attorney at the number listed below.

In view of the foregoing, it is believed that the remaining claims now distinguish over the prior art and are allowable. For the reasons discussed above, it is believed that this Application is now in an allowable condition such that it is appropriate to hereby respectfully solicit its allowance.

Respectfully submitted,

STEVEN G. SMARSH, ET AL.
CARGILL & ASSOCIATES, P.L.L.C.



Lynn E. Cargill
Registration No. 31,598
56 Macomb Place
Mt. Clemens MI 48043-5636
Phone: 586-465-6600

Date: September 9, 2003

C:\TheTextAP-3rdAOAResp030903

Merriam-Webster

- Home
- Help
- Word of the Day
- Word Games
- Word for the Wise
- Books and CDs
- Online Education
- Company Info
- Customer Service
- Network Options
- Language Zone
- The Lighter Side
- Site Map

Merriam-Webster Unabridged
Member Log-in

Merriam-Webster DICTIONARY

Atlas | Reverse Dictionary | Rhyming Dictionary
Dictionary | Thesaurus | Unabridged Dictionary

One entry found for **compiler**.

Main Entry: **com·pil·er**
Pronunciation: k&m-'pI-l&r
Function: *noun*
Date: 14th century
1 : one that compiles
2 : a computer program that translates an entire set of instructions written in a higher-level symbolic language (as Pascal) into machine language before the instructions can be executed

Get the Top 10 Most Popular Sites for "compiler"

For More Information on "compiler" go to Britannica.com

Find Photos, Magazines and Newspaper Articles about "compiler" at eLibrary. Free registration required.

Pronunciation Symbols
Click on the example word to hear it pronounced.

\&\ as a and u in abut	\e\ as e in bet	\o\ as aw in law
\ᵊ\ as e in kitten	\E\ as ea in easy	\oi\ as oy in boy
\&r\as ur/er in further	\g\ as g in go	\th\ as th in thin

Writing custom data to executable files in Windows and Linux

Abstract: *Think you can't tweak with your project after it's compiled? Check out these useful techniques for adding custom data to your EXE in Win32 and Linux. By Daniel Polistchuck.*

The code for this article is available at CodeCentral. Find it here

Writing custom data to EXE files in Windows and Linux

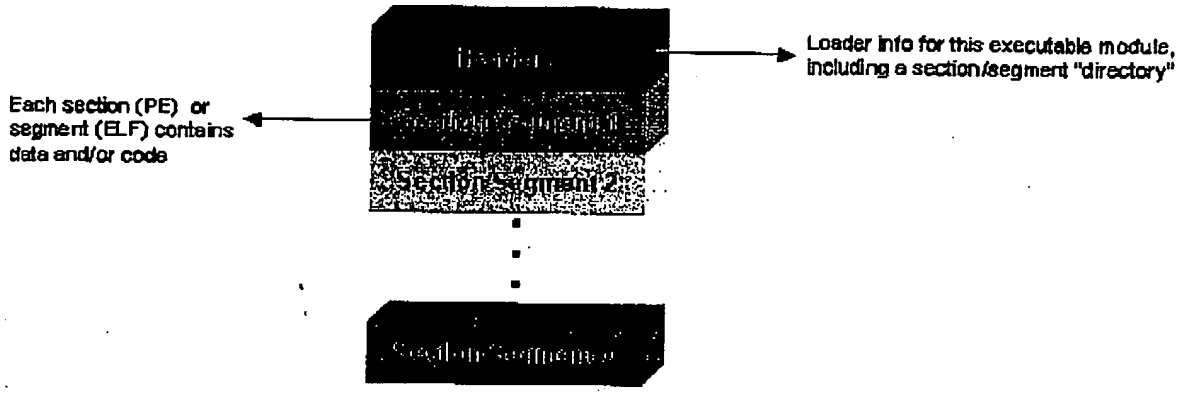
We developers are used to treating the executable code generated by our development tools as "black boxes." Nevertheless, there are times when the ability to write and read custom data to and from executable files would be really useful. Some interesting solutions can make use of the technique described in this article, including:

- software Licensing
- software customization
- security info
- post-compilation/linking addition of data to programs
- stubs
- Installation tools

How are executable files organized?

Each operating system defines its own executable-file internal format. We will be dealing here with Linux, which uses the industry-standard ELF (Executable and Linking Format) format and with Win32 operating systems that implement the Microsoft PE (Portable Executable) standard. Although the two formats differ in their inner structures, each can be seen as a self-contained, internally-referenced "file system." Both formats define a header that is divided into several parts. Some of these parts (the Section Header in PE and Program Header in ELF) work much like file allocation tables in FAT file systems: They describe the internal offsets to chunks of data and code in the executable file.

Simplified view of PE (Win32) and ELF (Linux, etc.) File Formats

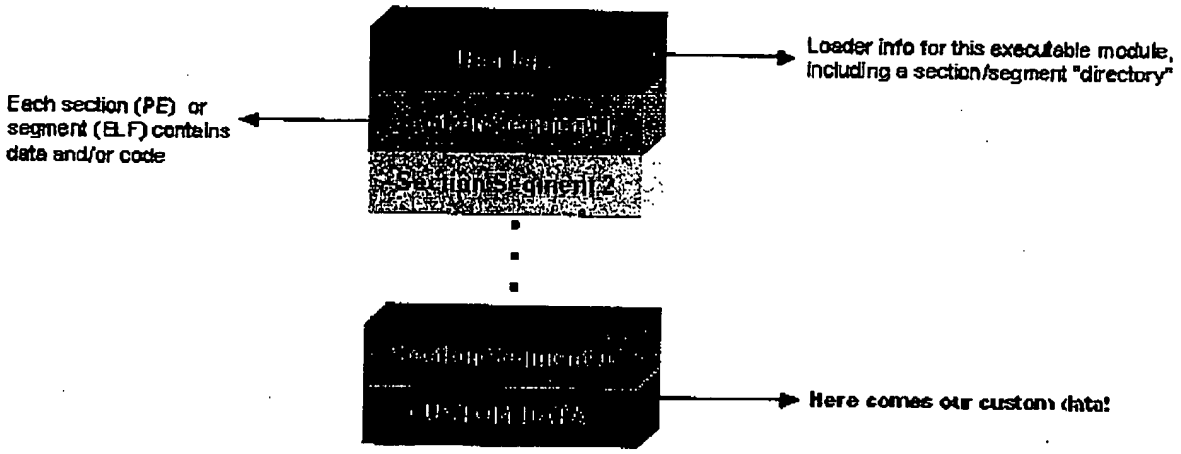


When a user or shell activates a file that is recognized as an executable file in Linux or Win32, the "loader" springs into action. The loader is responsible for loading the file from the disk, mapping it to memory, interpreting the file internal structures (headers and such), and starting the execution of the executable initial entry-point.

Where do we put the custom data?

Take a look at the picture above. Bearing in mind that the executable file works like a *self-containing* file system that is read from its beginning to its end, it is safe to assume that whatever is put *beyond* the end of the executable file won't compromise the executable information. So the best place to put custom data is after the EOF of the original executable file.

Adding our Custom Data to the Executable File



The Win32 and Linux Loaders happily ignore our CUSTOM DATA chunk.

So we can write anything to the executable?

Yes, that's correct. We can write anything past the EOF of an executable file in Linux and Win32. We are free to define any kind of structure and deal with it in our code any way we want to.

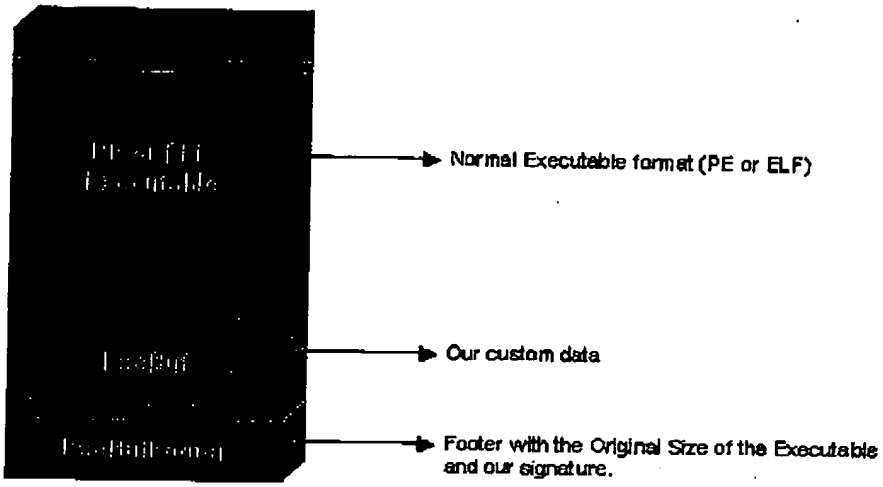
So let's add a footer to our executable file that will include some useful information:

```
const
  //our ExeBuffer signature
  ExeBufSig = 'EB1.0';

type
  //the Footer for our executable format
  TExeBufFooter = record
    OriginalSize : Integer;
    Sig : Array[0..4] of char;
  end;
```

- The ExeBufSig is a constant that will let us identify our "custom executable format."
- The TExeBufFooter is a record (yes, a record!) that will help us locate the signature and the custom data we add to the executable.

Our Custom Exe Format



How do we add custom data to the executable?

Linux uses POSIX file semantics, which lets us modify, delete, and rename a file that is already in use and not locked. But Win32 doesn't let us do that kind of stuff! It's all very complicated, but the bottom line is that Linux lets us write self-modifying executables, but Win32 doesn't.

To deal with that situation, we must write a second executable (the Writer), that will be responsible for writing custom data into the "client" (Reader) executable.

The code that is responsible for writing the data is simple:

```
procedure SetExeData (ExeName : String; ExeBuf : TExeBuf);
var
  F : File;
  BufSz,OrigSz : Integer;
  Footer : TExeBufFooter;
begin
  AssignFile (F,ExeName);
  Reset (F,1);
  try
    //obtaining the original file size
    OrigSz := FileSize(F);
    //go to the EOF of the file
    Seek (F,OrigSz);
    //Writing our custom data beyond the EOF
    BufSz := Length(ExeBuf);
    BlockWrite (F,Pointer(ExeBuf)^,BufSz);
    //Writing our footer
    FillChar (Footer,SizeOf(Footer),0);
```

```
    Footer.OriginalSize := OrigSz;
    Footer.Sig := ExeBufSig;
    BlockWrite (F,Footer,Sizeof(Footer));
  finally
    CloseFile (F);
  end;
end;
```

Simple, right? We treat the executable like any other file.

How do we read data back from the executable?

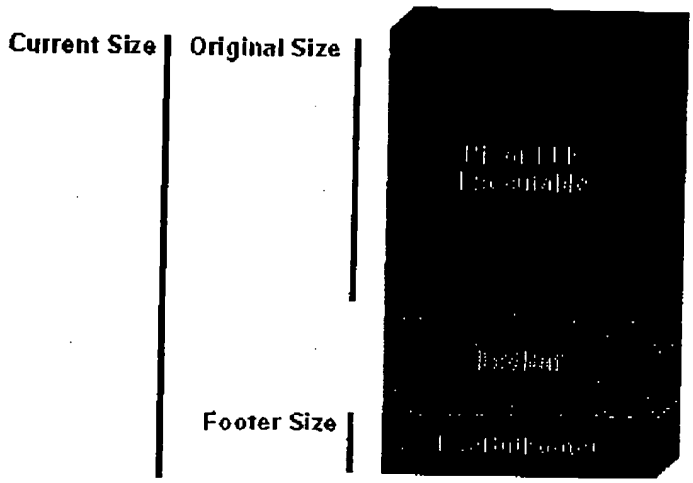
Everything we need to read the data is in the TExeBufFooter record located in the end of the executable. The exact location of the the footer is FileSize - SizeOf(TExeBufFooter). Upon reading the footer from the executable, we can obtain the original file size by accessing its OriginalSize field. So we can implement GetExeData like this:

```
procedure GetExeData (ExeName : String; var ExeBuf : TExeBuf);
var
  F : File;
  CurrSz, BufSize : Integer;
  OldFileMode : Integer;
  Footer : TExeBufFooter;
begin
  AssignFile (F,ExeName);
  //Saving the old FileMode
  OldFileMode := FileMode;
  //Setting the FileMode to ReadOnly
  FileMode := 0;
  try
    Reset (F,1);
    try
      //Getting the current file size
      CurrSz := FileSize (F);
      //Seeking to the footer position
      //and reading it
      Seek (F,CurrSz-SizeOf (Footer));
      BlockRead (F,Footer,Sizeof(Footer));
      //if there's no signature, boom!
      //no data in this executable!
      if Footer.Sig <> ExeBufSig then
        raise EExeBuf.Create ('No Data in EXE!');
      //calculating the buffer size that was written
      //to this executable file
      BufSize :=CurrSz-Footer.OriginalSize-SizeOf(Footer);
      SetLength (ExeBuf,BufSize);
      //seek and read!
      Seek (F,Footer.OriginalSize);
      BlockRead(F,Pointer(ExeBuf)^, BufSize);
    finally
      CloseFile (F);
    end;
```

```
    finally
      //returning to the previous saved
      //FileMode
      FileMode := OldFileMode;
    end;
  end;
```

This code may seem a little complicated at first glance, but the only real complication is the part where we calculate the buffer size (BufSize). In fact, it's very simple: We subtract from the current file size the original size and the footer size. Algorithms don't get much simpler than that.

Calculating the Buffer Size



ExeBuf Size = Current Size – Original Size – Footer Size

Now we throw in a couple of helper functions to convert TExeBuf to and from Strings:

```
procedure StringToExeBuf (const S : String; var ExeBuf : TExeBuf);
begin
  SetLength(ExeBuf,Length(S));
  Move (Pointer(S)^,Pointer(ExeBuf)^,Length(S));
end;

function ExeBufToString (const ExeBuf : TExeBuf) : String;
begin
  SetLength (Result,Length(ExeBuf));
  Move (Pointer(ExeBuf)^,Pointer(Result)^,Length(ExeBuf));
end;
```

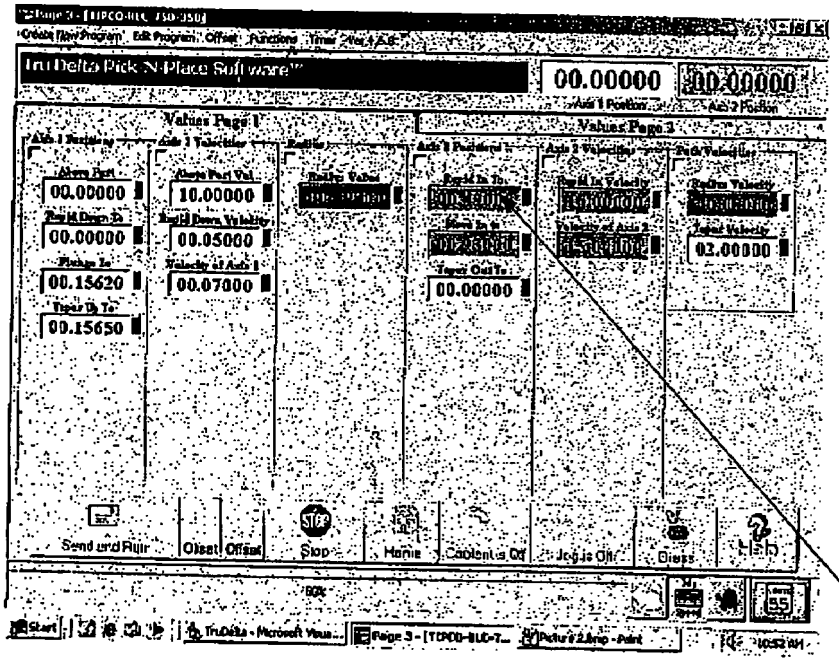
And that's it!

Daniel is the IT Director of QualTech IT and is eagerly waiting for the December 19 LOTR Release. He can be reached through e-mail at danpol@pobox.com.

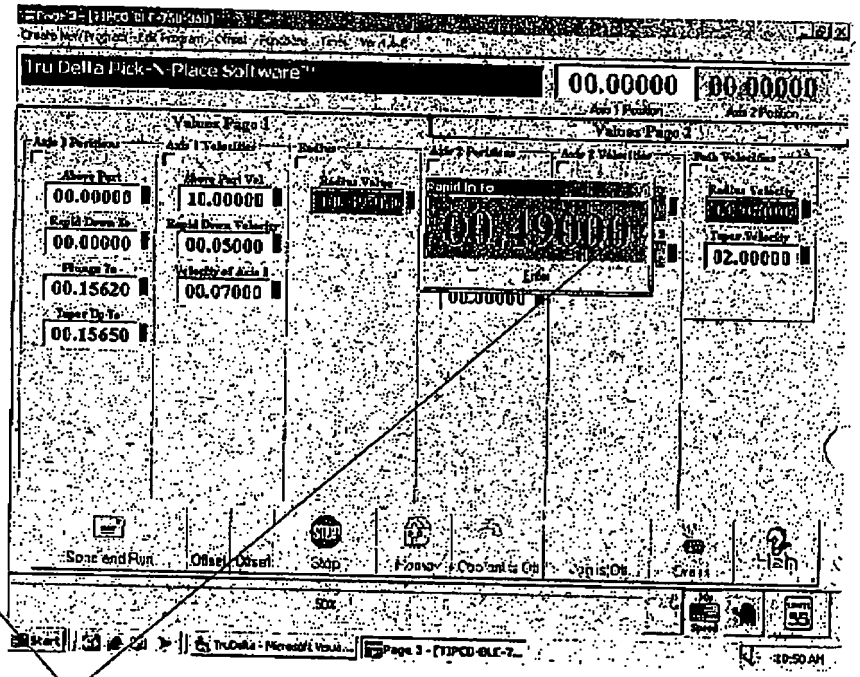
Add or View comments on this article

Article ID: 27979 Publish Date: November 26, 2001 Last Modified: November 26, 2001

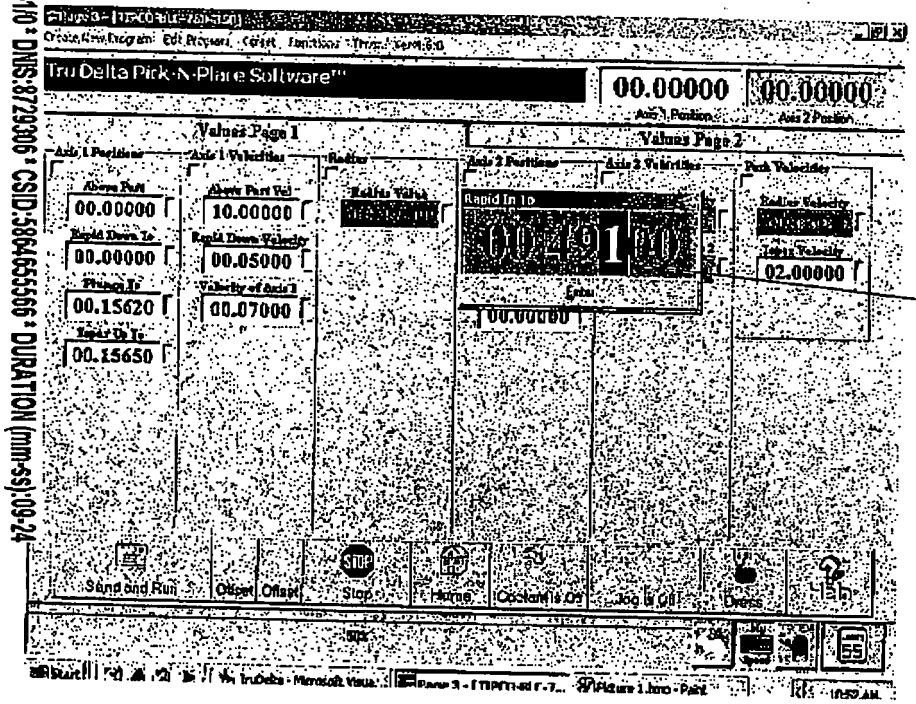
PROGRAM CREATION START
SELECT TEMPLATES
DEFAULT DATA ENTERED FROM DATABASE
USER MAKES CHANGES WHERE NECESSARY
PROGRAMMING COMPELETE
NO
YES
USER DATA STORED TO DATABASE
COMPILE
CREATION OF MOTION SCRIPT (USER DATA COMBINED WITH MACHINE CONSTANTS)
SCRIPT DOWNLOADED TO MOTION CONTROLLER
END PROGRAM CREATION
new way
MOST CNC GRINDERS REQUIRE A CNC PROGRAMMER TO UNDERSTAND G CODE. THIS PROGRAM USES NO G CODE. IT IS A PROPRIETARY SCRIPT LANGUAGE MADE DESIGNED BY COMPUMOTOR. HOWEVER THAT SCRIPTING LANGUAGE IS STILL TOO COMPLICATED FOR A NORMAL MACHINE OPERATOR TO UNDERSTAND. TRU TECH CREATED TEMPLATES FOR THE USER TO FILL OUT THEN THE COMPUTER CREATES A PROGRAM FOR THE MOTION CONTROLLER IN THE 6000 SERIES SOFTWARE LANGUAGE. IT COMPILES THE TEMPLATES INTO A SCRIPT AND SUBSTITUTES CHARACTERS WITH USER VALUES AND MACHINE CONSTANTS.
THESE ARE TEMPLATES FOR SCRIPTS
USER DATA
Tru Tech Program Creator
CREATE PROGRAM BUTTON
Plunge To
Velocity of Axis 1
Axis 1 Backlash
Return To Zero
00.15620
00.07000
00.00000
YES
old way
G CODE EDITOR
LOW LEVEL SCRIPT LANGUAGE
USES KEYBOARD ONLY NO MOUSE INPUT



MAIN PAGE WITH VALUES DISPLAYED



USER CLICKS ON VALUE AND A POP UP VALUE BOX DISPLAYS



USER THEN EDITS THE VALUES BY SCROLLING UP OR DOWN WITH THE MOUSE OR INPUTS THE VALUE WITH THE KEYBOARD

PAST MACHINES USE A TEXT FILE TO EDIT INDIVIDUAL VALUES. THIS ALLOWS A ENLARGED VIEW OF THE VALUE. THE VALUE BOX ALSO HAS LIMITS BUILT IN TO PREVENT ERRORS AND CRASHES.

CARGILL & ASSOCIATES
56 MACOMB PL
MOUNT CLEMENS, MI 48043

5323

WCMA® Working Capital Management™ Account

DATE 9-9-03

PAY TO THE ORDER OF Commissioner of Patents & Trademarks $ 580.00

Five Hundred Eighty & 00/100 DOLLARS

FOR Merrill Lynch

BANK ONE

MEMO 09/720,576 TruTech P-302

Lynn E Cargill

⑈005323⑈ ⑆044000804⑆ 01408254 3790⑈

The "Received" Stamp of the Patent & Trademark Office imprinted hereon acknowledges the filing of:

1. Transmittal Letter (orig & copy) - 2 Pgs.
2. Request For Continued Examination (RCE) Transmittal - 1 Pg.
3. Petition for Extension of Two (2) Months Time - 1 Pg.
4. Amendment - 10 Pgs., plus 10 Pgs. of Attachments (20 total)
5. Check No. 5323 for $580.00

DESCRIPTION OF PAPER:

NAME OF APPLICANT: Steven G. Smarsh, et al.

TruTech P-302

INTF. OR SERIAL NO: 09/720,576

The "Received" Stamp of the Patent & Trademark Office imprinted hereon acknowledges the filing of:

1. Transmittal Letter (orig & copy) - 2 Pgs.
2. Request For Continued Examination (RCE) Transmittal - 1 Pg.
3. Petition for Extension of Two (2) Months Time - 1 Pg.
4. Amendment - 10 Pgs., plus 10 Pgs. of Attachments (20 total)
5. Check No. 5323 for $580.00

DESCRIPTION OF PAPER:

NAME OF APPLICANT: Steven G. Smarsh, et al.

TruTech P-302

INTF. OR SERIAL NO: 09/720,576

RECEIVED SEP 18 2003 TECHNOLOGY CENTER R3700

OIPE SEP 12 2003 PATENT & TRADEMARK OFFICE

DATE: 9-9-03 ATTY: LEC